dexyfex.com

Galaxia: The basics – Coordinates

Posted on **July 11, 2016July 11, 2016** by **dexyfex**

Galaxia Demo July '16 - Voxel asteroids, landing…

So, let's start at the beginning.
All virtual worlds need some kind of a coordinate system to represent positions in. Most 3D games and simulations represent a position with 3 floating-point numbers representing the X, Y and Z distance from the world's origin (which is located at X,Y,Z=0,0,0). These numbers are usually in a meaningful unit such as meters, so a value of 27.4 might mean 27.4m. Which is all fine, for most games and simulations.

But any experienced programmer will tell you that there's a bit of a problem when the numbers start to get big. *Floating point precision* is a limiting factor, because the numbers are made up of a limited number of bits. Most engines currently use 32-bit floating point numbers (aka single precision, or "float") to represent almost everything, and currently available consumer GPUs are only really efficient with 32-bit operations. These single precision floats generally have about 7 significant figures to work with. For example, I should be able to fairly safely use numbers with as much detail as 123.4567, or 4.567891×10^74. Note how the position of the final digit of precision depends on the exponent value – so in the second case, the difference between incremental numbers is on the order 10^70! This can become a major problem when trying to do calculations involving both big and small numbers.

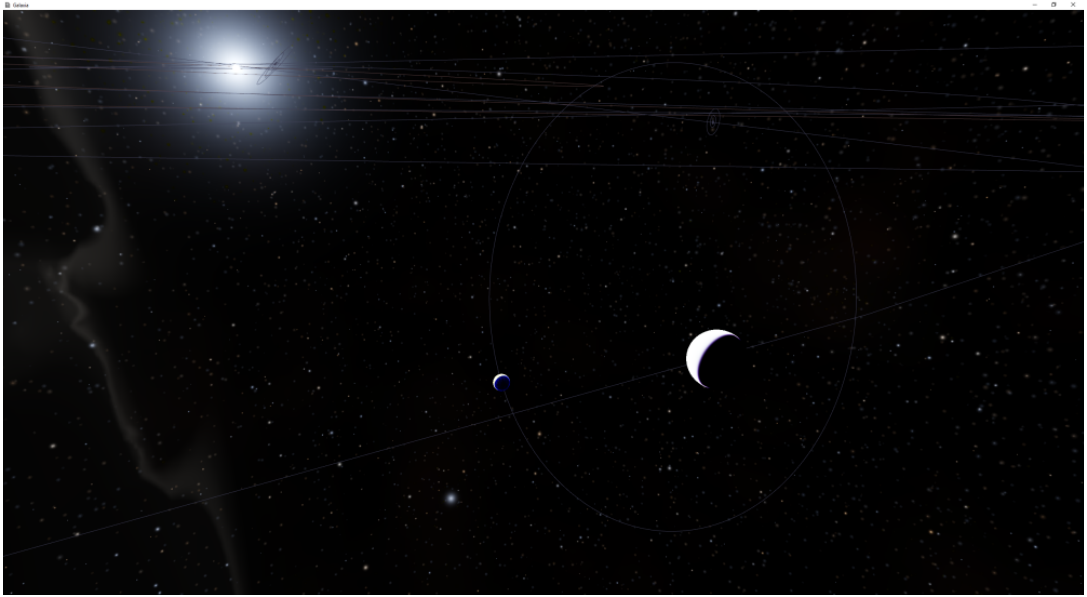
The problem can be somewhat reduced by using more bits. 64-bit floating point numbers (aka double precision or "double") are becoming more widely used due to native support from 64-bit processors. Obviously, using doubles to store position information will allow for much larger distances from the world origin before the calculations start to go wrong due to precision errors. But the problem is still there – and in doubles you just have doubled the amount of significant figures you have to work with. There's still that issue of having to be close to the world origin, with loss of numerical detail further out. And when a value *is* close to the origin, there's *too much* precision, being a waste of those precious bits.

Galaxia instead uses 64-bit integers. This allows all possible coordinate values to have the same resolution throughout a volume of space, meaning the least significant bit of each number always represents the same distance. A scaling factor just needs to be chosen to correspond integer units to real-world units, for example 10000 units per meter (=0.0001 meters per unit) might be used, meaning we could store positions accurate to 0.1mm. Doing the calculations on that to find the largest possible distance that can be stored:
2^63 / 10000 = 922,337,203,685,477.5808 meters
= ~1 trillion km (at 0.1mm resolution).
Which seems pretty big…

But still, calculations need to be done on objects at any point in the space, for example rendering a 3D object generally involves transforming object/vertex positions with a camera view-projection matrix to determine positions on the screen. The solution is to use *relative* positions when doing calculations. In the case of rendering in Galaxia, the camera is always assumed to be at position (0,0,0), meaning the camera's view matrix does not have a translation component. Before any object's screen position is calculated with the camera matrix, its position is calculated relative to the camera. In maths terms, this is:
rp = op – cp

where rp is the position relative to the camera, op is the object's position in the world, and cp is the camera's position. In programming terms, since op and cp are both 64-bit integer vectors, subtracting them as such will give an exact answer, and if the two positions are close to each other, the resulting numbers will be small. This means we can safely convert the relative position to a floating point format for use in the rendering calculations.

1 trillion kilometers might sound like a lot but it really is nothing on a galactic scale. So we still need some further method to be able to store those really gigantic values that will allow us to fly in between stars and galaxies. There's also a problem if we think about using one huge coordinate system to store positions of things on a planet's surface, for example. When the planet rotates and moves along in its orbit, we don't want to have to recalculate the position in "intergalactic coordinates" of everything on the surface every time, because there just wouldn't ever be enough computing power.



So in the tradition of killing two birds with one stone these two problems are solved with one solution – nested coordinate systems. Starting from the largest scale, we need intergalactic coordinates. Since truly vast distances have to be covered, a large scaling factor is required. 10^16 meters per unit is used in Galaxia, which allows for quite smooth motion relative to galaxies and results in a universe that stretches out to what might seem like infinity to the player (okay if you really want to know it's 9.22×10^34 meters, or 9.75×10^18 light years – many of orders of magnitude bigger than our observable universe).

Galaxies are positioned in the intergalactic universe space (obviously) by means of a pseudo-random number generator (but that's a different subject for another day). Along with the position, a size value and a random orientation quaternion are also generated (a quaternion is basically 4 floats). These values are then used to form the first "child" coordinate system – the galactic, or interstellar coordinates, which have a coordinate scale of 10000 meters per unit. That scale allows for a galaxy to be up to 9.75 million light years (mly) in radius, where the biggest known galaxy at this time is 5.8 mly in radius. For comparison, the Milky Way galaxy is only about 100,000 light years across (0.05 mly radius).

When the player flies close to a galaxy, a coordinate transform needs to occur. The player's position in the galactic (interstellar) coordinate system is calculated from the position in universal (intergalactic) coordinates, by first calculating the player position relative to the galaxy. Since this will be a small number in universe scale, we can then convert it to a floating point number. A unit conversion also needs to be performed, since galactic coordinates are on a different scale to universal coordinates. The easiest way to think about that is to first convert the number from universal units into meters (multiply by the universe meters per unit), and then convert back into galactic units (divide by the galaxy meters per unit). Once the relative position is found in galactic units, to find the final position in the galactic coordinates we need to rotate the vector by the inverse of the galaxy's orientation (using an inverse of the galaxy's quaternion), to account for the orientation of the galaxy relative to the universal coordinates. Essentially the same conversion is also done with the player's velocity and orientation, to make the transition seamless.

Once the position in galactic coordinates is obtained, the player's character object (which currently has no mesh in Galaxia) is moved out of the universal coordinate system's container and into the galactic one. Basically that means from now on, the player is moving around within the galaxy simulation rather than moving around in the outer universe simulation. In some senses this may sort of equate to an "instance" as the term is understood in online gaming – it's an isolated coordinate system that isn't really affected by others (except that you probably wouldn't need to ever have two of the same galaxy "instance" for example). The only main difference here is that even though the player moves around in the galaxy's coordinate system, they are still actually within the bounds of the universe coordinates, therefore the player now has two sets of coordinates – the primary one that is being controlled (the galactic) and the secondary (universal), which is calculated from that, using the reverse process as before (rotate by the galaxy quaternion and then adjust the scale).

The coordinate system nesting is continued in this manner for star systems. Objects like planets inside star systems again have their own coordinate systems within that. Currently the maximum coordinate system hierarchy depth in Galaxia is 5 (asteroids in asteroid belts), but there is really no limit to how far it could go, except for the potential increase in the amount of rendering needing to be done at each level.

When it comes to drawing everything, rendering happens from the outer coordinate systems to the inner ones, and since the player's position and orientation is calculated for each outer system, they are each rendered as normal. The primary depth buffer is also cleared after rendering each level if it's been used. To help with the problem of increasing rendering complexity at each level, a performance improvement is made at the star level by rendering everything that is seen outside the system to a skybox texture (i.e. the other galaxies, the current galaxy, and other stars). This does unfortunately cause a slight delay when entering a star system for the first time, since rendering the skybox requires rendering everything 6 times with different view orientations, for the 6 faces of the skybox. But once that skybox is rendered, we no longer have to worry about rendering all the galaxies etc. every frame while inside that star system.

When we get to the planet coordinate system, objects on the planet surface will have positions represented relative to the center of the planet. So when the planet moves and rotates, their positions can stay the same, meaning we don't have to re-position anything but the planet itself (or rather its coordinate system).

The main drawback to these nested coordinate systems is obviously that the bigger ones have lower unit resolution than the smaller ones. This may not be as big an issue as it first seems, since generally space is extremely empty. The main problem case might arise in a multiplayer scenario, if two players approach each other in the intergalactic space, for example. Since units are each 10^16 meters in that space, it would be impossible to have one player "standing" next to the other in that coordinate system. But the simple solution here is just to create a new coordinate system or set of coordinate systems as they approach each other, allowing the resolution of the distance between the two players to increase as they get closer to each other. In that multiplayer scenario that would actually correspond to creating an "instance" for that area of space where the two players can interact.

In addition to all this, Galaxia also has a permanent "local" coordinate system centered on the player character, which remains isolated from the other coordinate systems. This system is the one used for the camera controls – arc rotation around the player, zooming, and will allow for VR head tracking. I also find it interesting that due to the nature of floating point numbers, the full camera zoom range is quite adequately represented by a single 32-bit float.